PATENT

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

First Named Inventor:	Donald Robert Syme	Attorney Docket No.:	180610.01
Application No.:	10/025,270	Group Art Unit	2192
Filed:	December 18, 2001	Examiner:	Yigdall, Michael J.
Customer No.:	22971	Confirmation Number:	5667
Title: EFFICIENT GENERIC CODE IN A DYNAMIC EXECUTION ENVIRONMENT			

APPEAL BRIEF

To: Commissioner for Patents
PO Box 1450
Alexandria, Virginia 22313-1450

Dear Sir,

Pursuant to 37 C.F.R. §41.37, Applicant hereby submits an appeal brief for application 10/025,270, filed December 18, 2001, within the requisite time from the date of filing the Notice of Appeal. Accordingly, Applicant appeals to the Board of Patent Appeals and Interferences seeking review of the Examiner's rejections.

Appeal Brief Items	Page

Real Party in Interest

The real party in interest is Microsoft Corporation, the assignee of all right, title and interest in and to the subject invention.

Related Appeals and Interferences

Appellant is not aware of any other appeals, interferences, or judicial proceedings which will directly affect, be directly affected by, or otherwise have a bearing on the Board's decision to this pending appeal.

Status of Claims

Claims 1-14, 25 and 26 stand rejected and are pending in the Application. Claims 1-14, 25 and 26 are appealed. Some of these claims were previously amended. Claims 15-24 and 27-36 were previously canceled without prejudice. Claims 1-14, 25 and 26 are set forth in the Appendix of Appealed Claims on page 22.

Status of Amendments

An Office Action was issued on December 12, 2004.

A Response to the Office Action was filed June 2, 2005. Claims 1, 25 and 26 were amended. Claims 15-24 and 27-36 were cancelled without prejudice.

A Final Office Action was issued on August 25, 2005.

A Response to the Final Office Action and a Request for Continued Examination was filed on January 25, 2006. Claims 1, 25 and 26 were amended.

An Office Action was issued on March 31, 2006.

Appellant filed a Notice of Appeal on September 9, 2006 in response to the Advisory Action and the Final Office Action.

Summary of Claimed Subject Matter

The pending independent claims are claims 1, 25, and 26. A concise explanation of each of the independent claims is provided below.

Claim 1 is directed to a computer process for dynamically generating typing context data associated with a typing-context-relevant-code-point being executed within a typing context in a dynamic execution environment. The computer process comprises encountering the typing-context-relevant-code-point in the typing context during execution of the program (page 26, lines 14-15), identifying a typing context handle associated with the typing context, the typing context handle referencing a typing context data structure associated with the typing context (page 26, line 21 and page 27, lines 14-15 and page 28, lines 10-11 and page 29, lines 2-3), computing the typing context data associated with the typing-context-relevant-code-point (page 27, lines 1-2 and page 27, lines 17-18 and page 28, lines 12 - 15 and page 29, lines 9-11), dynamically allocating a field in the typing context data structure associated with the typing-context-relevant-code-point (page 26, lines 6-7 and page 27, line 23 - page 28, line 1 and page 28, lines 19-20 and page 29, lines 15-16), the field describing the exact type of the typing-context-relevant-code-point in the typing context (page 8, lines 17-19), and recording the typing context data in the field of the typing context data structure (page 26, lines 7-8 and page 28, lines 1-2, and page 28, lines 20-21 and page 29, lines 16-17).

Claim 25 is directed to an execution engine for execution engine for executing parametrically polymorphic code and dynamically generating typing context data associated with a typing-context-relevant-code-point being executed within a typing context in a dynamic execution environment. The execution engine comprises a read module configured to encounter the typing-context-relevant-code-point in the typing context during execution of the program (page 26, lines 14-15), a handle module configured to identify a typing context handle associated with the typing context, the typing context handle referencing a typing context data structure associated with the typing context (page 26, line 21 and page 27, lines 14-15 and page 28, lines 10-11 and page 29, lines 2-3), a computation module configured to compute the typing context data associated with the typing-context-relevant-code-point (page 27, lines 1-2 and page 27, lines 17-18 and page 28, lines 12 - 15 and page 29, lines 9-11), an allocation module configured to dynamically allocate a field in the typing context data structure associated with the typing-context-relevant-code-point (page 26, lines 6-7 and page 27, line 23 - page 28, line 1 and page 28, lines 19-20 and page 29, lines 15-16), the field describing the exact type of the typing-context-relevant-code-point in the typing context (page 8, lines 17-19), and a recording module configured to record the typing context data in the field of the typing context data structure (page 26, lines 7-8 and page 28, lines 1-2, and page 28, lines 20-21 and page 29, lines 16-17).

Claim 26 is directed to a method of dynamically generating typing context data associated with a typing-context-relevant-code-point being executed within a typing context in a dynamic execution environment. The method comprises encountering the typing-context-relevant-code-point in the

typing context during execution of the program (page 26, lines 14-15), identifying a typing context handle associated with the typing context, the typing context handle referencing a typing context data structure associated with the typing context (page 26, line 21 and page 27, lines 14-15 and page 28, lines 10-11 and page 29, lines 2-3), computing the typing context data associated with the typing-context-relevant-code-point (page 27, lines 1-2 and page 27, lines 17-18 and page 28, lines 12 - 15 and page 29, lines 9-11), dynamically allocating a field in the typing context data structure associated with the typing-context-relevant-code-point (page 26, lines 6-7 and page 27, line 23 - page 28, line 1 and page 28, lines 19-20 and page 29, lines 15-16), the field describing the exact type of the typing-context-relevant-code-point in the typing context (page 8, lines 17-19), and recording the typing context data in the field of the typing context data structure (page 26, lines 7-8 and page 28, lines 1-2, and page 28, lines 20-21 and page 29, lines 16-17).

Grounds of Rejection to be Reviewed on Appeal

Claims 1-14 stand rejected under 35 U.S.C. § 101 because the claimed invention is directed to non-statutory subject matter.

Claims 1-5, 7-9, 25 and 26 stand rejected under 35 U.S.C. §102(b) as being anticipated by Viroli et al., *Parametric Polymorphism in Java through the Homogeneous Translation LM: Gathering Type Descriptors at Load-Time* (hereinafter "Viroli").

Claims 6 and 10-14 stands rejected under 35 U.S.C. §103(a) as being unpatenable over Viroli in view of U.S. Patent 5,093,914 to Coplien et al. (hereinafter "Coplien").

Argument

I. The rejection under 35 U.S.C. §102(b) over Viroli is improper because Viroli does not anticipate each and every element of the claims.

Claims 1-5, 7-9, 25 and 26 stand rejected under 35 U.S.C. §102(b) as being anticipated by Viroli.

Applicant respectfully submits that the Office has not established a proper anticipation rejection because Viroli does not anticipate each and every element of the claims.

A. The §102 Standard

In making out a §102 rejection, the Federal Circuit has stated that "[a] claim is anticipated only if each and every element as set forth in the claim is found, either expressly or inherently described, in a single prior art reference." *Verdegall Bros. v. Union Oil Co. of California*, 814, F.2d 628, 631, 2 USPQ2d 1051, 1053 (Fed. Cir. 1987). Furthermore, "[t]he identical invention must be shown in as complete detail as is contained in the...claim...." *In re Bond*, 910 F.2d 831, 15 USPQ2d 1566 (Fed. Cir. 1990).

B. Discussion of Parametric Polymorphism

The field of object oriented programming is generally directed to defining classes and creating objects, or, instances of classes. A class generally includes methods, which may be called to perform an operation, and properties or other

data containers which may be accessed to either get or set data within an object, or, instance of a class. Most object oriented programming languages include other features, however, for the purposes of discussion only these few features will be discussed.

Classes may inherit from a parent class; such a parent class may be commonly referred to as a base class. An inherited class may include all the inheritable methods, properties, and other data structures associated with the base class. An inherited class may extend the base class by adding other methods, properties, and other data structures. An inherited class may also extend the base class by overriding methods of the base class. Methods of the base class are overridden when the inherited class defines a method with the same name and parameter list as the method on the base class.

A method on a base class that may be overridden is known as a "polymorphic" method. Polymorphism in object oriented programming refers to the ability of a method to take "many forms". Consider for example a base class defined for an "animal class" that includes methods, properties and data structures for defining the characteristics common to all animals. The animal class may include a method called "makeNoise()" that may include instructions to play a sound common to all animals.

Also consider a class that may be defined for "dog class" that inherits from the animal class. The dog class inherits all the methods, properties, and other data structures of the base class. The dog class may override the makeNoise() method by implementing a method with the same name and including the same parameters. In order to "morph" the method specifically for the dog class, the makeNoise() method may include instructions to play a

barking sound. When a “dog object” is created using the dog class, the makeNoise() method may be called and the proper sound of a dog barking will be played. That is, the makeNoise() method has morphed and executes differently to exhibit the characteristics of a dog.

Now consider a software tool created to play animal sounds. Such a software tool may be written such that the tool only includes code to instantiate animal objects from the animal class. The tool may then call the makeNoise() method of each object to play the animal sound. The tool may use the dog class to instantiate a dog object because the dog object inherits from the animal class; that is, the dog class includes everything necessary to instantiate an animal class. The tool may then call the makeNoise() method on the dog object and the makeNoise() method may play the correct barking sound.

Polymorphism may be illustrated more clearly when a “cat class” is considered. The cat class may inherit from the animal class and override the makeNoise() method to play a sound of a cat. The tool may instantiate the cat class to create a cat object and then call the makeNoise() method to play the sound of a cat. That is, the method has morphed to exhibit the characteristics of a cat. Because the discussed tool includes code to instantiate animal objects, the tool does not need to include code to instantiate specific animal objects such as a dog or cat object; such objects “just work” with the tool due to their inheritance from the animal class.

Next, consider the case in which a method on a base class includes typed parameters. The makeNoise() method is not shown as including a typed parameter, however, methods may commonly take typed parameters such an integer or a string. If a method in a base class includes one or more typed

parameters, an inheriting class must define the overridden method using the exact same one or more typed parameters. Such a requirement is necessary during compilation to ensure that a creator of an inherited class has not made an error in overriding the method. In particular, a class may include a method of the same name more than once provided each method includes a different set of typed parameters.

Consider a class that has been designed to perform operations in a generic way. For example, such a class may include methods such as “Add” and “Subtract”. An Add method may add the contents of two variables together. A Subtract method may subtract the contents of two variables. Inherited classes may be derived from such a base class for specific types such as an integer, a string, and the like. However, in order for the Add and Subtract methods to be polymorphic, these methods may have necessarily been defined on the base class with the appropriate type. That is, if an inherited class is created to perform operations in a generic way on integers, the base class would have been required to define an “Add(int a, int b)” method and the inherited class may then have overridden this specific Add method. If an inherited class is created to perform operations in a generic way on strings, the base class would have been required to define an “Add(string a, string b)” method and the inherited class may then have overridden this specific Add method.

As can be seen from this example, the base class must contemplate all the various types that will be used by inherited classes in the future, a less than ideal situation. Consider now a “generic” type that may be defined in the “Add” method of the base class. Such a generic type may indicate that a type may be any legal type. For example, the Add method may now be defined on the base

class as “Add(<T> a, <T> b)”. When an inherited class inherits from the base class and defines the “Add” method, the inherited class may define the type that “<T>” represents. For example, the inherited class may define <T> to represent an integer, a string, or the like. When an object is instantiated from the inherited class, the generic type <T> is replaced with the specific type.

The substitution of actual types for generic types may be performed by a compiler when code is compiled. However, in a dynamic execution environment such as Microsoft’s .Net Frameworks and the Java runtime, code may not be compiled until executed. Substitution of actual types may therefore not be performed until the code is compiled during execution and is therefore not a simple substitution operation.

C. Summary of Viroli

Viroli discloses a parametric polymorphism solution for the Java language. Consider that the Java runtime does not natively support parametric polymorphism; many different workaround solutions have been proposed for the Java runtime. Viroli is based on the “GJ” (Generic Java) proposal. In Generic Java, a generic type such as <T> may be defined that is not actually a generic type, but rather is a reference to the Java type “Object”. The Java type “Object” is the root of the inheritance hierarchy for Java; that is, all types in Java inherit from “Object”. A separate compile-time translation process then “erases” or down-casts the “Object” type to create the actual type. In Generic Java, all generic types are translated using such a technique at run-time. Viroli extends Generic Java by utilizing the “reflection” functionality of Java to delay translation

of the "Object" type to actual types until the "Object" is encountered during execution, also known as "load-time".

D. Summary of Key Differences Between Application and Viroli

As previously discussed, Viroli allows the definition of generic types which are initially implemented as the Java "Object" type that are then down-cast to translate the "Object" to the specific type. Viroli extends the Generic Java system by including a hash table to cache already instantiated objects to improve performance.

In contrast, the present application makes use of a "typing context" for generic types. More particularly, the compiler may create an execution context during runtime. At this time, the compiler may encounter a "typing context" relevant code point, or, a generic open type. The compiler determines the operation being performed with the generic type, then obtains a typing context handle for the current execution context. The typing context handle references a dynamically allocatable runtime type descriptor that describes the exact type of the generic type in the current typing context and execution context. The runtime descriptor may then be used to substitute the exact runtime type for the generic type.

E. Claim 1

Claim 1 is directed to a computer process comprising encountering the typing-relevant-code-point in the typing context during program execution, identifying a typing context handle associated with the typing context with the typing context handle referencing a typing context data structure, computing

the typing context data associated with the typing-relevant-code-point, dynamically allocating a field in the typing context data structure with the field describing the exact type of the typing-context-relevant-code-point, and recording the typing context data in the field.

In applying Viroli to the application, the Office broadly interprets several elements of Viroli to be equivalent to the presently claimed subject matter. However, it must be noted that the disclosure of Viroli does not disclose a specific implementation of a parametric polymorphism system; that is, Viroli discusses a possible implementation of a parametric polymorphism system but does not disclose the internal mechanisms by which such an implementation may be completed.

In particular, the disclosure of Viroli discloses sample code for use in the proposed pararmetric polymorphism system. More particularly, the parametric polymorphism system of Viroli is described in the context of how sample code would be executed in the system. Therefore, the parametric polymorphism system of Viroli is not specific and the specific internal structures of Viroli can not be extrapolated from sample code for use in the system and a general discussion of the functioning of the sytem.

The lack of specificity of the internal structures by which the parametric polymorphism system of Viroli may be implemented prevents an overly broad application of Viroli to the claimed subject matter of the application by the Office. For example, the Office asserts that Viroli discloses "identifying a typing context handle associated with the typing context, the typing context handle referencing a typing context data structure associated with the typing context" at page 12, section 4.2, first paragraph of Viroli. The Office asserts that the

type descriptor manager, type descriptors, and hash tables of Viroli are equivalent to the claimed subject matter (see page 5, Office Action of 3/31/2006).

However, even interpreted as broadly as possible, the cited section of Virioli does not disclose "identifying a typing context handle associated with the typing context." In particular, the Office has failed to explain with specificity how the cited section of Viroli is being applied to the claimed matter. The cited section of Viroli does not recite a typing context, a typing context handle, or the action of identifying a typing context handle associated with the typing context. In contrast, the cited section of Viroli discloses "[t]o summarize, LM translation is supported by three library classes" and a discussion of what is included in the three library classes.

A possible application of Viroli to the claimed subject matter may be the broad interpretation of the typing context data structure as the "hash table" of Viroli. Again, such a broad interpretation is not proper as the only commonality is that a hash table may be a data structure. Regardless, the cited section of Viroli does not disclose that a hash table is included in the system, rather, Viroli discloses that the parametric polymorphism system merely includes methods to allow its correct handling <u>by</u> hash tables.

Therefore, it is not possible for Viroli to describe further operations depending on the identifying operation as Viroli fails to disclose the identification of a typing context handle associated with the typing context and a typing context data structure associated with the typing context. The Office does attempt to equate "computing the typing context data associated with the typing-context-relevant-code-point" with the "translation" of Viroli disclosed at

page 6, section 3, and "instantiating parametric types" of Viroli disclosed at page 11, section 4.1, first paragraph (see Office Action of 3/27/2006, page 5).

Once again the Office fails to state with specificity which elements of Viroli are performing the functionality of the claimed subject matter. In particular, the cited sections of Viroli do not disclose "typing context data", let alone "computing typing context data". In particular, the "translation" at page 6, section 3 of Viroli does not involve the computation of typing context data, and is in fact the discussion of a possible approach to parametric polymorphism that is ultimately rejected by Viroli. See Viroli, page 8, section 3, last sentence, "[o]ur target then was to develop a translation based on the same mechanism, but which was able to bound run-time overhead at acceptable values".

Furthermore, the very fact that the Office asserts that "translation" is applicable to the presently claimed subject matter demonstrates the overly broad interpretation of Viroli. As discussed earlier, the "translation" of the instantiated objects demonstrates that the "exact type" is not used; rather, Viroli discloses the use of "translated objects". Viroli does not disclose that a translated object is a type, let alone an exact type; and even if Viroli were to disclose a translated object being equivalent to an type a translated object is <u>translated</u> and not exact.

Regardless, it is not clear how "instantiating parametric types" anticipates "computing the typing context data associated with the typing-context-relevant-code-point". That is, the Office is attempting to broadly interpret "instantiating parametric types", however, "instantiating parametric types" is a function of every programming language and is not specific to Viroli. That is,

consider that any type used as a parameter must be instantiated before it may be used.

In advancing the rejection the Office does specifically state that page 11, section 4.1, third paragraph of Viroli shows "dynamically allocating a type descriptor that describes an exact type such as 'Cell<Integer>'". However, the Office is in error when stating that "Cell<Integer>" describes an "exact type". As can be seen from the earlier discussion of parametric polymorphism, "Cell<Integer>" is an object instantiated from the "Cell" class with an "Integer" substituted for the generic open-type parameter. An "Integer" is an exact type; and Viroli is silent with regard to a type descriptor being an exact type such as an "Integer".

Rather, the cited section of Viroli is directed to the registration of "type descriptors", or, instances of objects with substituted actual types such as "Integer" in a hash table. The purpose of registering such type descriptors is for the purpose of efficiency; Viroli discloses that a hash table is used to "prevent a given instantiation of a parametric type having two ore more type descriptors living at run-time" (see Viroli, page 11, section 4.1, first paragraph, first sentence).

Finally, the Office refers to the same cited section of Viroli in asserting Viroli discloses "recording the typing context data in the field of the typing context data structure", stating Viroli "shows recording the type descriptor in a hash table" (see Office Action of 3/27/2006, page 6). The application of the exact same section to two separate elements of the claimed subject matter is notable because the Office is attempting to draw an equivalency between one element of Viroli and more than one element in the claimed subject matter.

That is, the Office attempts to draw an equivalency between "a field in the typing context data structure" and a "type descriptor" of Viroli and an identical equivalency between "typing context data" and a "type descriptor" of Viroli.

This attempt to draw broad equivalencies between the disclosure of Viroli and the claimed subject matter is indicative of the overly broad interpretation of Viroli by the Office. As has been established, Viroli does not disclose a typing context handle or a typing context, let alone a typing context associated with the typing context handle. Therefore, it follows that since Viroli does not disclose these elements, Viroli may not disclose further operations performed using a typing context.

For at least the above-identified reasons, applicant respectfully submits that claim 1 is not anticipated by Viroli and is allowable. Accordingly, the rejection of claim 1 should be withdrawn.

F. <u>Claims 2-5, 7-9</u>

Claims 2-5 and **7-9** depend from claim 1 and are allowable at least by virtue of that dependency. Accordingly, the rejections of claims 2-5 and 7-9 should be withdrawn.

G. <u>Claims 25 and 26</u>

Claim 25 stands rejected because of the Office's assertion that the execution engine recited in the claim corresponds to the computer program product of claim 1. The Applicant respectfully disagrees the execution engine recited in claim 25 corresponds to the computer program product of claim 1 and does not acquiesce to such a characterization of the claimed subject matter

of claim 25. However, as the Office has failed to provide a substantive rejection of claim 25, the Applicant respectfully requests the rationale discussed with respect to the rejection of claim 1 be applied to the present rejection to claim 25.

Claim 26 stands rejected because of the Office's assertion that the method recited in the claim also corresponds to the computer program product of claim 1. The Applicant respectfully disagrees the execution engine recited in claim 26 corresponds to the computer program product of claim 1 and does not acquiesce to such a characterization of the claimed subject matter of claim 26. However, as the Office has failed to provide a substantive rejection of claim 26, the Applicant respectfully requests the rationale discussed with respect to the rejection of claim 1 be applied to the present rejection to claim 26.

II. The rejection under 35 U.S.C. §103(a) over the combination of Viroli and Coplien does not establish a *prima facie* case of obviousness.

Claims 6 and 10-14 stand rejected under 35 U.S.C. §103(a) as being unpatentable over Viroli in view of U.S. Patent 5,093,914 to Coplien et al. (hereinafter "Coplien").

Applicant respectfully submits that the Office has not established a *prima facie* case of obviousness with respect to establishing that the prior art teaches or suggests all elements of the claims.

A. The §103 Standard

To establish a *prima facie* case of obviousness, three basic criteria must be met. First, there must be some suggestion or motivation, either in the references themselves or in the knowledge generally available to one of ordinary skill in the art, to modify the reference or to combine reference teachings. Second, there must be a reasonable expectation of success. Finally, the prior art reference (or references when combined) must teach or suggest all the claim limitations. The teaching or suggestion to make the claimed combination and the reasonable expectation of success must both be found in the prior art, and not based on applicant's disclosure. *In re Vaeck*, 947 F.2d 488, 20 USPQ2d 1438 (Fed. Cir. 1991).

B. Claims 6 and 10-14

Claims 6 and 10-14 depend from claim 1 and, therefore, include the elements recited in claim 1.

The Office asserts that Viroli teaches the computer system in claim 1 (see page 9, Office Action of 3/27/2006). However, contrary to this assertion, the Office never establishes that Viroli teaches or suggests all of the elements of claim 1, either alone or in combination with Coplien. In fact, in the Final Office Action issued on 3/27/2006, claim 1 stands rejected only under Viroli.

As discussed above, Viroli fails to disclose or suggest each element of claim 1. The Office never establishes that Coplien cures this defect in Viroli. Therefore, since claims 6 and 10-14 include the elements of claim 1 by virtue of its dependency, the Office has failed to establish a *prima facie* case of

obviousness for claims 6 and 10-14 with respect to establishing that the prior art teaches or suggests all elements of the claim.

Claims 6 and 10-14 are allowable over the cited references for at least this reason. Therefore, the rejections of claims 6 and 10-14 should be withdrawn.

III. The rejection to Claims 1-14 under 35 U.S.C. §101 should be withdrawn because the claimed invention is directed to statutory subject matter.

A. Claim 1.

Claim 1 recites a "computer program product encoding a computer program for executing on a computer system...." The Office Action points out that Applicant's specification defines "computer program product" as including, *inter alia*, "a computer data signal embodied in a carrier wave...." For this rejection, the Office Action has relied on the *Interim Guidelines for Examination of Patent Applications for Patent Subject Matter Eligibility* (1300 OG 142), Annex IV.

Applicant notes that said guidelines are merely interim in nature and that no official proceeding has established a rule that specifies that a computer data signal embodied in a carrier wave is not statutory subject matter. The Patent Office has issued patent applications with such language in the past and Applicant asserts that until an official finding is made as to the patentability of such subject matter, the subject matter is patentable subject matter. In fact, a case with such a claim has issued as late as a week before the filing of this appeal. *See, e.g.*: US Patent No. 7,139,874 (claims 40) <u>issued 11-21-06</u>; US Patent No. 7,134,021 (claim 13) issued <u>11-07-2006</u>; US Patent No. 7,139,874; US Patent No. 7,130,596 (claims 30-33) issued <u>10-31-2006</u>.

B. Claims 2-14.

Claims 2-14 depend from claim 1 and are allowable for at least the reasons discussed above. Accordingly, the rejections to claims 2-14 should be withdrawn.

Conclusion

The Office's basis and supporting rationale for the § 102(b) and § 103(a) rejections is not supported by the disclosure of the cited references. Furthermore, the Office's rejection under § 101 is now moot. The Applicant respectfully requests that the rejections be overturned and that the pending claims be allowed to issue.

Respectfully Submitted,

Dated: 11-29-2006

By: James R. Banowsky

James R. Banowsky Reg. #37,773
Peter Taylor

Microsoft Corp.
One Microsoft Way,
Redmond, WA 98052
425-705-3539

CERTIFICATE OF MAILING OR TRANSMISSION [37 CFR 1.8(a)]

I hereby certify that this correspondence and the documents identified on this form are being electronically deposited with the USPTO via EFS-Web on the date shown below:

November 29, 2006
Date

Noemi Tovar

Appendix of Appealed Claims

1. A computer program product encoding a computer program for executing on a computer system a computer process for dynamically generating typing context data associated with a typing-context-relevant-code-point being executed within a typing context in a dynamic execution environment, the computer process comprising:

encountering the typing-context-relevant-code-point in the typing context during execution of the program;

identifying a typing context handle associated with the typing context, the typing context handle referencing a typing context data structure associated with the typing context;

computing the typing context data associated with the typing-context-relevant-code-point;

dynamically allocating a field in the typing context data structure associated with the typing-context-relevant-code-point, the field describing the exact type of the typing-context-relevant-code-point in the typing context; and

recording the typing context data in the field of the typing context data structure.

2. The computer program product of claim 1 wherein the typing-context-relevant-code-point executes a type test on an instance of a generic class, the typing context data includes a resource type descriptor defining the exact type of the instance, and the computer process further comprises:

performing the type test based on the resource type descriptor associated with the typing-context-relevant-code-point.

3. The computer program product of claim 1 wherein the typing-context-relevant-code-point executes an allocation of an instance of a generic class, the typing context data includes a resource type descriptor defining the exact type of the instance, and the computer process further comprises:

creating the instance of the generic class based on the resource type descriptor associated with the typing-context-relevant-code-point, wherein the instance is of the exact type.

4. The computer program product of claim 1 wherein the typing-context-relevant-code-point calls a generic method, the typing context data includes another typing context handle, and the computer process further comprises:

passing the other typing context handle referencing the typing context data to the generic method as a hidden parameter.

5. The computer program product of claim 1 wherein the identifying operation comprises:

retrieving the typing context handle from a stack frame.

6. The computer program product of claim 1 wherein the typing-context-relevant-code-point is executed within an instance of a generic class and the identifying operation comprises:

retrieving a first pointer to the instance; and

retrieving the typing context handle via a second pointer, a second pointer being relative to the first point and referencing the typing context handle associated with the instance.

7. The computer program product of claim 1 wherein the computing operation comprises:

retrieving the typing context data associated with the typing-context-relevant-code-point from a global hash table.

8. The computer program product of claim 1 wherein the encountering operation comprises:

assigning an index to the typing-context-relevant-code-point.

9. The computer program product of claim 8 wherein the allocating operation comprises:

allocating the field in the typing context data structure, in accordance with the index.

10. The computer program product of claim 8 wherein the index is assigned based on the "arity" of the typing-context-relevant-code-point.

11. The computer program product of claim 8 wherein the index is assigned based on a category associated with the typing-context-relevant-code-point.

12. The computer program product of claim 11 wherein the category is assigned on a per-containing class basis.

13. The computer program product of claim 11 wherein the category is assigned on a per-containing method basis.

14. The computer program product of claim 11 wherein the category is assigned on a per-containing assembly basis.

15. - 24. Canceled

25. An execution engine for executing parametrically polymorphic code and dynamically generating typing context data associated with a typing-context-relevant-code-point being executed within a typing context in a dynamic execution environment, the execution engine comprising:

a read module configured to encounter encountering the typing-context-relevant-code-point in the typing context during execution of the program;

a handle module configured to identify identifying a typing context handle associated with the typing context, the typing context handle referencing a typing context data structure associated with the typing context;

a computation module configured to compute computing the typing context data associated with the typing-context-relevant-code-point;

an allocation module configured to dynamically allocate allocating a field in the typing context data structure associated with the typing-context-

relevant-code-point, the field describing the exact type of the typing-context-relevant-code-point in the typing context; and

a recording module configured to record recording the typing context data in the field of the typing context data structure.

26. A method of dynamically generating typing context data associated with a typing-context-relevant-code-point being executed within a typing context in a dynamic execution environment, the method comprising:

encountering the typing-context-relevant-code-point in the typing context during execution of the program;

identifying a typing context handle associated with the typing context, the typing context handle referencing a typing context data structure associated with the typing context;

computing the typing context data associated with the typing-context-relevant-code-point;

dynamically allocating a field in the typing context data structure associated with the typing-context-relevant-code-point, the field describing the exact type of the typing-context-relevant-code-point in the typing context; and

recording the typing context data in the field of the typing context data structure.

27. - 36. Canceled

EVIDENCE APPENDIX

None

RELATED PROCEEDINGS APPENDIX

None